Modelling Program's Performance with Gaussian Mixtures for Parametric Statistics

Julien Worms, Sid Touati

Modelling Program's Performance with Gaussian Mixtures for Parametric Statistics

Julien WORMS and Sid TOUATI

Abstract—This article is a continuation of our previous research effort on program performance statistical analysis and comparison [1], in presence of program performance variability. In the previous study, we proposed a formal statistical methodology to analyse program speedups based on mean and median performance metrics: execution time, energy consumption, etc. However mean and median observed performances do not always reflect the user's feeling of such performances, especially when they are particularly unstable. In the current study, we propose additional precise performance metrics, based on performance modelling using Gaussian mixtures. Our additional statistical metrics for analysing and comparing program performances give the user more precise decision tools to select best code versions, not necessarily based on mean or median numbers. Also, we provide a new metric to estimate performance variability based on Gaussian mixture model. Our statistical methods are implemented with R and distributed as open source code.

Index Terms—Gaussian mixture, statistical testing, parametric statistics, program performance modelling, program performance variability, benchmarking.

1 INTRODUCTION

When someone reads books or articles on computer architectures, performance analysis, operating systems (OS) or compilation, he may still think that the execution time of a program P on a fixed machine M with fixed data input I is stable around a value, which can be denoted as by single number `ExecutionTime`(P,I,M). This situation was true for old computers, but nowadays nobody really observe constant execution times, except in rare situations: ideal execution environment, special processor architectures devoted to performance stability, bare mode execution, sudo or root access to control the OS, etc.

In everyday life of computer usage, everybody hardly ever observes constant execution times, even with fixed data input and low overhead operating systems workload. The consequence is that the reported values of program performances in the literature are not easily reproducible, and it becomes more and more difficult to select the most effective program version or OS configuration, especially on high performance multicore and manycore processors executing parallel applications.

When one considers a fixed binary code and input data, and such code is executed n times on exactly the same machine with the same OS with the fixed data input, n distinct performance measurements would be observed. Here, performance may be execution time, energy consumption, memory usage, network traffic, Instructions Per Cycle (IPC), Cycles Per Instruction (CPI), etc. Usually, execution time is the most important and interesting performance metric in practice, it is easily measured by OS commands or hardware performance counters. In any scientific physical experiment, performance measurement tools have naturally limited precision and sensitivity, so it is common to observe some variability as in any physical measurement. Such variability may be considered as noise due to non perfect measurement process. However, additional uncontrolled factors induce substantial variation in the observed performance. Here are below some known categories of such factors [2]:

Technological factors: variable clock frequency of a chip, variable Input/Output transfer times which depend on the hardware, asynchronous peripherals, etc.

Micro-architectural factors: Out of order execution mechanism of superscalar processors, hardware branch prediction, hardware data prefetching, memory hierarchy effects (multiple level of caches, some are shared between cores, some are private).

Software competition to access shared resources A parallel application may have multiple processes or threads that execute concurrently and compete for resources, such as common memories, busses, networks, etc.

Operating system factors: Process and thread scheduling policies, dynamic memory allocation, NUMA effects, etc.

Algorithmic factors: Some parallel programs implement algorithms that assign thread workload differently from one execution to another. So some threads have more or less workload from one execution to another, leading the synchronisation barriers (meeting points) to finish early or lately depending on the workload of the critical path thread. Also, some parallel algorithms are designed to be non deterministic, they behave differently from one execution to another.

All the above factors explain why it is difficult in practice to have stable performances. In some situations, when the variability factors are understood and known, and when the user has enough expertise and administration rights, one can reduce the variability as done in [3]. But in general, people usually cannot reduce such performance variability, or cannot easily understand the exact factors for every software on any combination of hardware machine and operating system configuration.

One could think that the variability of program performance is due to a sort of noise, and that the performance can be considered as an average value plus a noise term varying around zero (therefore, the performance value could be modelled as a Gaussian distribution, and reduced to its mean or median value). Indeed, this is not always true: this variability is very often not of that simple type. Indeed, many observed benchmarks exhibit multimodal density functions. Fig. 1 illustrates an example with the observed executions times of a SPEC benchmark (equake). It has been executed 1000 times on a dedicated single user Linux machine. The X axis represents the observed execution times in seconds, while the Y axis represents the frequency (histograms).

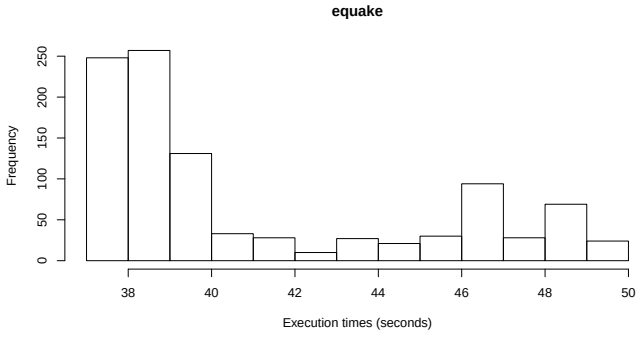

Figure 1: Observing the execution times of equake

When the distribution of program performances is multi-modal, the usage of simple indicators, such as the mean or median execution time, are useless because the performances are not grouped around a single number like in mono-modal distributions. We furthermore provided empirical evidence in [2] (by conducting statistical tests) that most of the observed programs performances are not only mono-modal, but also they are not of Gaussian nature. In theory, it means that some statistical tests designed for Gaussians (such as the Student t-test) would compute a wrong risk level. Likewise, the well known formula that computes the confidence interval of an average would correspond to a wrong confidence level. Of course, if the data sample is large enough, but nobody would be able to define how *large* it should be, the error must be asymptotically bounded and can be considered negligible.

The above introduction motivates our current research effort, which is a synthesis of a complete and long research report [2]. In the presence of performance variability, we must rely on formal statistics to select the best code version. In the past, we presented such statistical protocol called the Speedup-Test [1]. It analyses the mean and median execution time only. In the current work, we extend our previous study as follows:

1) We build a statistical modelling based on Gaussian mixtures to fit multi-modal performance data.
2) We build a statistical test to quantify the quality of data-model fitting.
3) We define new code performance metrics that go beyond mean and median performances.
4) For each new performance metric, we propose a parametric estimation based on Gaussian mixtures modelling.
5) We implemented all our statistical methods using R, and we demonstrated its practical efficiency.
6) Our software, called `VARCORE`, is publicly distributed as a free open source code.

Our article is organised as follows. We define and recall the notations of some basic statistical notions in Sect. 2 that will be used later. We study a new distribution model based on Gaussian mixtures (GM) in Sect. 3. We chose GM as a target data distribution model because we observed in practice that sample distributions are often multi-modal. Building GM models from data is called clustering, which we present in Sect. 4. In Sect. 5, we describe a statistical method which checks whether a GM model fits well some given experimental data. Then, based on GM modelling, we propose new program performance metrics based on parametric and non-parametric statistics in Sect. 6. A summary of our extensive experiments is given in Sect. 7. Some state of the art of program performance analysis using statistics is summarised in Sect. 8. Limitations and future research plan are presented in Sect. 9. Finally we conclude with a synthesis and some opinions.

2 Basic notations and definitions in statistics and probability theory

Let $X \in \mathbb{R}$ be a continuous random variable. Let $\mathcal{X} = (x_1, \ldots, x_n)$ be a sample of observations issued from the same distribution as X. The following items recall basic notations and definitions used in this artile. Below, x denotes some arbitrary real number.

Absolute value is noted $|x|$.

Indicator function is noted $\mathbb{1}$, it is defined by $\mathbb{1}_A = 1$ if the relation A holds true, and $= 0$ otherwise. For example, $\mathbb{1}_{x \leq y} = 1$ if $x \leq y$, and 0 otherwise.

Probability density function (PDF or p.d.f.) of the random variable X is noted f_X. This well-known function describes the relative likelihood that this random variable takes a value (it is often approximately presented as $f_X(x) \simeq \mathbb{P}\left[x \leq X \leq x + dx\right]/dx$)

Probability is noted $\mathbb{P}[.]$. For instance, for $x \in \mathbb{R}$, the probability that $X \leq x$ is $\mathbb{P}[X \leq x] = \int_{-\infty}^{x} f_X(t)\, dt$.

Probability under hypothesis is noted $\mathbb{P}_{H_0}[.]$. It is equal to the probability $\mathbb{P}[.]$ under the assumption that some hypothesis H_0 (about the distribution of X) is true.

Cumulative distribution function (CDF or c.d.f.) of the random variable X is the function noted F_X defined by $F_X(x) = \mathbb{P}[X \leq x] = \int_{-\infty}^{x} f_X(t)dt$.

Empirical distribution function (EDF) of the sample $\mathcal{X}$ is the function, noted $\widetilde{F}_{\mathcal{X}}$, built from the observations $\mathcal{X}$ and which estimates the true CDF F_X: for every $x \in \mathbb{R}$, $\widetilde{F}_{\mathcal{X}}(x)$ is defined as the proportion of the observations $x_1, \ldots, x_n$ which are lower or equal to the value x (it is a step function which jumps by $1/n$ every time it reaches one of the observations x_i, $i = 1, .., n$, until it finally equals 1)

The expected value (or theoretical mean) of X is noted $\mu_X = \mathbb{E}[X] = \int_{-\infty}^{+\infty} x f_X(x) dx$.

The sample mean of the sample $\mathcal{X}$ is noted $\overline{\mathcal{X}}$, it is equal to $\frac{1}{n}\sum_{i=1}^{n} x_i$.

The theoretical median of the variable X is noted $\text{med}(X)$, it satisfies $F_X(\text{med}(X)) = \frac{1}{2}$.

The sample median of the sample $\mathcal{X}$ is noted $\overline{\text{med}}(\mathcal{X})$.

The theoretical variance of the random variable X is noted $\sigma_X^2 = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right]$

The sample variance of the sample $\mathcal{X}$ is noted $s_{\mathcal{X}}^2$, it is equal to $\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{\mathcal{X}})^2$

The standard deviation of the random variable X is noted $\sigma_X = \sqrt{\sigma_X^2}$

The sample standard deviation of the sample $\mathcal{X}$ is noted $s_{\mathcal{X}} = \sqrt{s_{\mathcal{X}}^2}$

The Gaussian PDF is the p.d.f. of the Gaussian distribution $\mathcal{N}(\mu, \sigma)$, defined by $\varphi(x; \mu; \sigma) = (2\pi\sigma^2)^{-1/2} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$, which has expectation μ and standard deviation σ.

The standard Gaussian PDF is the p.d.f. of the standard Gaussian distribution $\mathcal{N}(0, 1)$, and it is denoted by the function $\varphi(x) = \varphi(x; 0; 1)$

The Gaussian CDF is the c.d.f. associated to the Gaussian distribution $\mathcal{N}(\mu, \sigma)$, it is denoted by the function $\Phi(x; \mu; \sigma) = \int_{-\infty}^{x} \varphi(t; \mu; \sigma)\, dt$

The standard Gaussian CDF is the c.d.f. of the standard Gaussian distribution $\mathcal{N}(0, 1)$, it is denoted by the function $\Phi(x) = \Phi(x; 0; 1)$

Other notations will be introduced later in the text (in particular notations related to Gaussian mixtures, estimations of their parameters, and the performance metrics).

The next section explains and defines the theoretical probability model based on Gaussian mixtures.

3 Gaussian mixtures

The central idea of our work stems from the following remark: according to our experience in parallel and sequential application performance analysis, the variability of such data is not necessarily of the "deviation around a single mean value" kind, it often exhibits a clear clustering pattern: in other words, the execution times often vary from each other by clustering around two or more central values. Therefore, we cannot choose for our modelling family classical families of distributions such as the Gaussian, Exponential, Gamma or Weibull family, which are by nature unimodal. And summarising the data by a single mean or median value can be misleading for the end-user, when the time comes to compare different program versions of code optimisation methods: he could miss important features of the performance distribution.

In practice, many parallel HPC applications executed on supercomputers or on HPC multicore machines exhibit such multi-modal execution times distributions, even if we keep the input data fixed and if we execute the application on a dedicated machine. This observation remains true even if we use different compilers and distinct code optimisation flags, and even if we run the application with different numbers of threads and affinity strategies. It is in practice difficult to obtain stable performances.

According to our practical studies, single unimodal distributions such as the Gaussian, Lognormal and Weibull distributions are bad general models. Consequently, we propose to model the execution times by *mixtures of Gaussian distributions*: this family of distributions has proved to be an essential tool in many areas of scientific activities for many years now (biology, engineering, astronomy, among many others), particularly in the image analysis and pattern recognition fields. Mixtures of Gaussian distributions is a highly flexible family of distributions which can fit a great variety of data: it is not only naturally adequate for modelling data which exhibits clusters, but it can also (to some extent) handle the problem of possible skewness in the data, despite the symmetry of the Gaussian components of the mixtures. In addition, this family of distributions can efficiently model multivariate data, but in this work we will limit ourselves to univariate data.

Remind that we consider data $\mathcal{X} = (x_1, \ldots, x_n)$ which are independent realisations of a probability density function (p.d.f.) f_X. We will say that the data are issued from a finite mixture of Gaussian distributions (or simply a Gaussian mixture, which we will abbreviate by GM) if f_X is equal to some p.d.f. $g_{\theta,K}$ (parametrised by θ and K described below) of the form:

$$\begin{aligned} g_{\theta,K}(x) &= \pi_1\varphi(x; \mu_1; \sigma_1) + \ldots + \pi_K\varphi(x; \mu_K; \sigma_K) \\ &= \textstyle\sum_{k=1}^{K} \pi_k\varphi(x; \mu_k; \sigma_k) \end{aligned} \quad (1)$$

where:

- $\pi_1, \ldots, \pi_K$ are the mixture weights, which are positive and sum to 1;
- $\mu_1, \ldots, \mu_K$ and $\sigma_1, \ldots, \sigma_K$ are the mean values and standard deviations of the mixture individual components;
- $\varphi(\,\cdot\,; \mu_k; \sigma_k)$ denotes the p.d.f. of the Gaussian/normal distribution $\mathcal{N}(\mu_k, \sigma_k)$;
- K is the (integer) number of components in this mixture;
- θ is a vector gathering all the parameters (except K) in a single notation,

$$\theta = (\pi_1, \ldots, \pi_K\,;\, \mu_1, \ldots, \mu_K\,;\, \sigma_1, \ldots, \sigma_K)$$

Examples of GM probability distributions are illustrated in Fig. 2: the plain line corresponds to $K = 3$ and $\theta = (0.5, 0.35, 0.15\,;\, 2, 8, 25\,;\, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$, the dashed line to $K = 3$ and $\theta = (\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\,;\, 4, 16, 23\,;\, 1.5, 2, 2)$, and for the dotted line $K = 2$ and $\theta = (0.35, 0.65\,;\, 10, 14\,;\, 2, 4)$.

We will denote by $\mathcal{F}^{\text{GM}}$ the set of all mixtures of Gaussian distributions, and say that X is GM-distributed if its cumulative distribution function (c.d.f.) F_X belongs to $\mathcal{F}^{\text{GM}}$, which means there exists some parameters K and $\theta = (\pi_1, \ldots, \pi_K\,;\, \mu_1, \ldots, \mu_K\,;\, \sigma_1, \ldots, \sigma_K)$ such that

$$\forall x \in \mathbb{R},\ F_X(x) \text{ equals } F_{\theta,K}(x) = \sum_{k=1}^{K} \pi_k \Phi(x; \mu_k; \sigma_k) \quad (2)$$

which is itself equivalent to f_X being equal to the density $g_{\theta,K}$ defined in Equ. 1. Naturally, the more components the mixture has, the more flexible the shape of the distribution

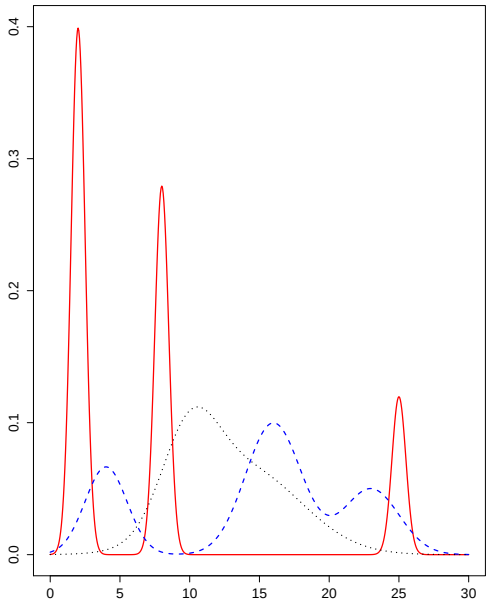

Figure 2: Examples of Gaussian mixtures distributions

can be (but this has a cost: the model has more parameters to be estimated).

The next section presents the method for building a GM model based on a data sample. This is called clustering in the literature.

4 Clustering method

In this section, we explain how the Gaussian mixture model is concretely estimated from the data. A detailed explanation would require many mathematical details and prerequisites, so we decided to provide only an overview of the principle and concepts at stake, so that the reader can grasp the main idea and understand the issues of the crucial step: the choice of an adequate number K of components in the Gaussian mixture.

Our aim is thus, for the moment, to estimate the parameters of the distribution described in Equ. 1, from which we assume our data are issued: the parameters are the clusters weights $(\pi_k)_{k=1..K}$, the clusters means $(\mu_k)_{k=1..K}$ and the clusters standard deviations $(\sigma_k)_{k=1..K}$, and they are gathered in one single notation, θ (which is $3K$-dimensional). The use of this estimated GM model for providing concrete statistical insight about the data at hand will be the subject of Sect. 6. For the moment, the most important is to consider the clusters means as the central values around which the program performances tend to cluster (execution times for instance).

4.1 Estimation of the parameters of the mixture for fixed K

How then is θ estimated ? Remind that the dimension of θ depends on the value of K, which is fixed for the moment. We naturally adopt a parametric approach, and intend to compute the maximum likelihood estimator $\hat{\theta}$ of θ, which is defined as the (global) maximizer of the log-likelihood function. This function, given the observations $x_1, \ldots, x_n$ and Equ. 1, is defined by:

$$L(\theta) = \ln \prod_{i=1}^{n} g_{\theta,K}(x_i) = \sum_{i=1}^{n} \ln \left(\sum_{k=1}^{K} \pi_k \varphi(x; \mu_k; \sigma_k) \right)$$

Maximising it consists in calculating the various partial derivatives of L with respect to the different parameters π_k, μ_k, σ_k $(k = 1, \ldots, K)$, and equalising them to 0 to obtain the so-called score equations. It is rather clear that any attempt to directly solve these score equations will turn out to be an unsolvable problem, due to the presence of a log of a sum.

This major obstacle was overcome thanks to the approach synthesised in the celebrated *EM algorithm* [4]. The acronym EM means a succession of E-steps (E for Expectation) and M-steps (M for Maximisation), which should lead to obtaining (numerically) the value of the maximum likelihood estimator $\hat{\theta}$. The starting idea of the EM algorithm is to presume that there exists, for each data value x_i, an unobserved label C_i with values in $\{1, \ldots, K\}$ which is the number of the cluster to which the data value x_i is most likely associated. For instance, a data value of 39.1 in the equake example of Section I, would most likely be associated to the first cluster on the left. If these labels $C_1, \ldots, C_n$ were observed along with the data sample $x_1, \ldots, x_n$ itself, then this would lead to another expression of the log likelihood function L (then called *complete likelihood*), much simpler to maximise (than the *incomplete likelihood* defined above), and which would lead to a simple computation of the maximum likelihood estimator $\hat{\theta}$.

The EM algorithm then runs like this. Starting from an initial guess $\theta^{(0)}$ of $\hat{\theta}$, the E-step generates labels $C_1, \ldots, C_n$ which are coherent with the data values and the current value $\theta^{(0)}$. Then the M-step computes the new value $\theta^{(1)}$ as the maximum likelihood estimator associated to the complete likelihood and to the labels generated in the E-step. It is proved that iterating this process necessarily increases the likelihood, *i.e.* at the j-th step of the algorithm, the value $L(\theta^{(j)})$ is necessarily higher than the previous one, $L(\theta^{(j-1)})$. The algorithm runs until the gain $L(\theta^{(j)}) - L(\theta^{(j-1)})$ is considered as negligible, and $\hat{\theta}$ is then defined as the last computed value $\theta^{(j)}$.

This general description of the EM algorithm in a clustering context turns out to greatly simplify when considering Gaussian mixtures: if we introduce the following notations, for $j \geq 0$,

$$\theta^{(j)} = (\pi_1^{(j)}, \ldots, \pi_K^{(j)}, \mu_1^{(j)}, \ldots, \mu_K^{(j)}, \sigma_1^{(j)}, \ldots, \sigma_K^{(j)})$$

and, for $i = 1, \ldots, n,\ k = 1, \ldots, K,\ j \geq 1$,

$$\alpha_{i,k}^{(j)} = \frac{\pi_k^{(j-1)} \varphi(x_i; \mu_k^{(j-1)}; \sigma_k^{(j-1)})}{\sum_{l=1}^{K} \pi_l^{(j-1)} \varphi(x_i; \mu_l^{(j-1)}; \sigma_l^{(j-1)})}$$

then we have the following simple formulas relating $\theta^{(j-1)}$

to $\theta^{(j)}$: for every $k = 1, \ldots, K$

$$\begin{aligned}
\pi_k^{(j)} &= \frac{1}{n} \sum_{i=1}^{n} \alpha_{i,k}^{(j)} \\
\mu_k^{(j)} &= \sum_{i=1}^{n} \frac{\alpha_{i,k}^{(j)}}{\sum_{i'=1}^{n} \alpha_{i',k}^{(j)}} \, x_i \\
\sigma_k^{(j)} &= \left(\sum_{i=1}^{n} \frac{\alpha_{i,k}^{(j)}}{\sum_{i'=1}^{n} \alpha_{i',k}^{(j)}} \left(x_i - \mu_k^{(j)} \right)^2 \right)^{1/2}
\end{aligned}$$

These formulas will not be proved here, see the book [5] for justifications (but more self-contained proofs, specific to this Gaussian mixtures case, can be found without too much difficulty in academic course notes).

4.2 Determination of the number K of components of the mixture

Now that the estimation of θ has been explained for a given value of K, let us explain how the number K of components of the Gaussian mixtures model can be chosen. In practice, several candidate values are considered for K, and one of them, noted $\hat{K}$, is chosen so that the corresponding GM model best fits the data at hand. The determination of $\hat{K}$ is nearly the most important issue in clustering analysis, and in this work we will adopt a simple and widespread strategy: using the BIC criterion (BIC stands for Bayesian Information Criterion).

The principle of the BIC criterion for determining $\hat{K}$ is the following. For a given $K \geq 1$, we note L_K the maximum value of the log likelihood for the model with K components (or more precisely, the maximum value which is issued from the EM algorithm, which is hopefully the actual maximum likelihood). Then it should be easy to conceive that the greater K is, the greater the value L_K will be: indeed, for instance, if we consider the model with $K+1$ components which best fits the data, then it will certainly better fit the data than the best model having K components (maybe not far better, but better all the same). Therefore, choosing K which maximises L_K will not work. The likelihood value needs to be penalised by a value which grows with K, in order to counterbalance the fitting gain that more complex models yield. That is the idea of the so-called information criterions, for instance the BIC criterion: to *choose the value of K that minimises the value* $BIC(K) = -2L_K + K \ln(n)$ (this value of the penalisation term $K \ln(n)$ has theoretical justifications, which will not be detailed here). Therefore, if $BIC(\hat{K}) = \max_{K \geq 1} BIC(K)$, then the model with $\hat{K}$ components will be a tradeoff between good data fitting and reasonable complexity. Note that, in practice, the maximum is chosen among a finite number of candidate values, for instance $1 \leq K \leq K_{max}$ (where K_{max} does not exceed 10 in general).

From now, $\widehat{F}_{\mathcal{X}}^{\mathrm{GM}}$ will denote the Gaussian mixture distribution $F_{\hat{\theta},\hat{K}}$ estimated from the observed sample $\mathcal{X}$, following the procedure we described above. This notation is important, since we will also deal in Sect. 5 with GM distributions estimated from samples which are different from $\mathcal{X}$.

4.3 Simple examples

Let us consider two well known SPEC benchmarks which are named `galgel` and `apsi`. Running these benchmarks multiple times on the same low overhead machine results in variable execution times, as illustrated by the histograms in Fig. 3. These histograms show the frequency of observed execution times (samples of 35 runs). After clustering, the approximate theoretical models based on GM are illustrated with the curves in Fig. 3. As can be seen, the execution times of these benchmarks exhibit multi-modal behaviour.

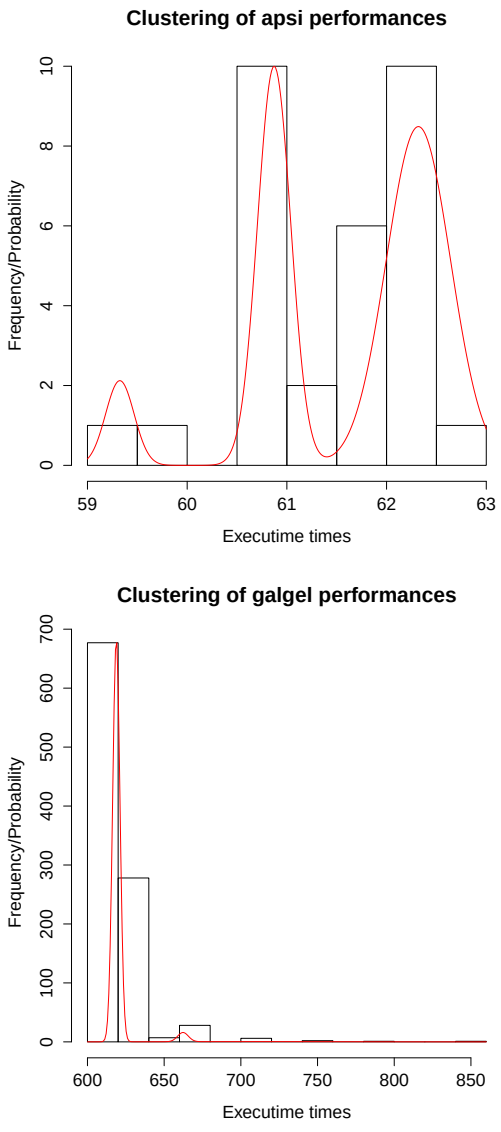

Figure 3: Gaussians mixture modelling

However, how can we check whether this computed GM models fit the data well or not? Usual statistical studies simply accept a graphical validation by visualising the match between the CDF of the model and the CDF of the sample: if they are close enough each other, then we conclude that the modelling is satisfactory. We draw in Fig. 4 the empirical CDF (step staircase function) versus theoretical CDF (continuous curve). For both examples, the

fitting seems fairly good graphically, but it is not formally evaluated.

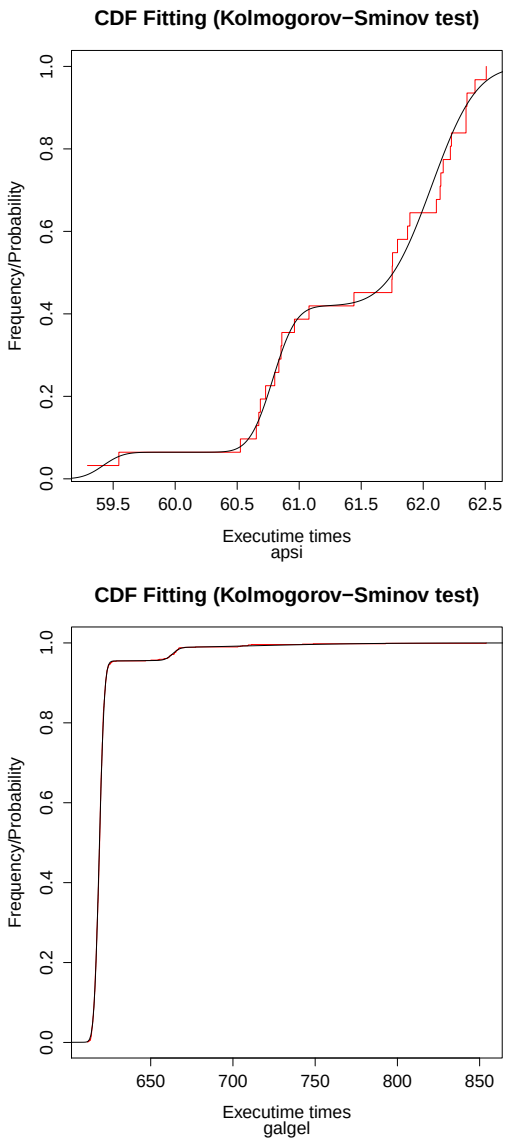

Figure 4: Quality of data fitting (empirical vs. theoretical CDF)

In our study, we do not rely only on graphical validation. We developed a complete automatic statistical test to check whether the model fits the data well or not. The next section is devoted to this issue.

5 Checking the Fitting of the Data to the Gaussian Mixture Model

We address in this section the important problem of validating our modelling program performance by Gaussians mixtures. This problem, known in the statistical domain as "assessing goodness-of-fit of the model" is here motivated, explained and addressed in details in the report [2]. We thus consider a method for validating the modelling of performance data by Gaussian mixtures (based on a usual goodness-of-fit metric), and the use of the *bootstrap method* for concretely calibrating this validation method is one of the main contribution of our work.

5.1 Description of the method

5.1.1 Preliminaries about goodness-of-fit

In the previous section, we explained how the parameters of the Gaussian mixture are estimated from the sample data via the EM algorithm. However, so far we did not address an important issue: what if the program performances are not issued from a Gaussian mixture distribution, but from another family of distributions ?

Let us be clear from the start: since we will rely upon statistical methods, we cannot be *absolutely sure* that our data are GM-distributed, we can only build a procedure which assesses, with some given level of confidence, whether the data are *not* fitting the GM family of distributions. It is important to understand that a given dataset may reasonably fit a handful of distributions families: all that we want is to build a reliable statistical test which will warn us when the Gaussian mixture model is a bad model for our data. With such a tool, we will be in a position of assessing that a very large amount of execution times data can be reasonably modelled by a Gaussian mixture.

The statistical test we are about to describe is called, in the statistical language, a *goodness-of-fit test*. The most famous of this kind of statistical tests is the χ^2-goodness-of-fit test, which tests whether some given discrete/integer data fits some given probability distribution. We need here a test which applies to continuous data: our goodness-of-fit test will be based on the computation of a distance between the data and the GM distribution estimated from the data. The distance we chose is the well-known Kolmogorov-Smirnov distance between the empirical distribution function $\widetilde{F}_X$ and the estimated GM distribution function $\widehat{F}_X^{\text{GM}}$:

$$KS_{\mathcal{X}} = ||\widetilde{F}_{\mathcal{X}} - \widehat{F}_{\mathcal{X}}^{\text{GM}}||_\infty = \sup_{x\in\mathbb{R}} \left| \widetilde{F}_{\mathcal{X}}(x) - \widehat{F}_{\mathcal{X}}^{\text{GM}}(x) \right| \quad (3)$$

where[1]

$$\widetilde{F}_{\mathcal{X}}(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{x_i \leq x} \quad \text{and} \quad \widehat{F}_{\mathcal{X}}^{\text{GM}}(x) = \sum_{k=1}^{\hat{K}} \hat{\pi}_k \Phi(x; \hat{\mu}_k; \hat{\sigma}_k)$$

The underlying idea is the following: if the data are issued from a distribution which is too different from (or cannot be approached by) a GM distribution, then it is clear that the distance $KS_{\mathcal{X}}$ will be large, *i.e* larger than the distance which would be computed if the data actually came from a GM distribution.

Our goal is then to define a statistical test of the hypothesis:

- H_0: the underlying distribution F_X of the data is a GM distribution (which means H_0: "$F_X \in \mathcal{F}^{\text{GM}}$"), against the alternative hypothesis
- H_1: the underlying distribution F_X of the data does not belong to the family of Gaussian mixture distributions.

In practice, *we will reject H_0 in favour of H_1 (with a given risk α) if the distance $KS_{\mathcal{X}}$ is too large, i.e. if $KS_{\mathcal{X}}$ exceeds some*

1. the ∞ symbol in subscript of $\|$ in Equ. (3) is a common notation in statistics, meaning that the distance is computed by taking the supremum over all $x \in \mathbb{R}$.

given critical value $c = c_\alpha$.

Of course, the crucial point in statistical testing is: what does "large" mean ? what is the risk α associated with a given choice of the value c ? For example, for a sample of size $n = 30$, what is the probability of falsely rejecting H_0 when the chosen value is $c = 0.15$? Answering this question is called *calibrating* the test, *i.e.* to be able to determine a critical value $c = c_\alpha$ associated with a given risk α (which is the risk of rejecting H_0 when the data is actually distributed as a Gaussian mixture). For instance, if for $n = 30$ we have $\mathbb{P}_{H_0}[KS_{\mathcal{X}} > 0.189] \simeq 0.05$, then a choice of $c = 0.189$ will provide a test of risk 5%, and a choice of $c = 0.15$ will provide a test with risk greater (or much greater) than 5%.

Generally, this calibration is made possible by a mathematical theorem, valid under some conditions (often including conditions on the size of the sample n). For instance, concerning the usual Kolmogorov-Smirnov (KS) test, which tests H_0: "$F_X = F_0$" for some *unique* and given distribution F_0, a famous theorem (see Theorem 19.3 [6]) states that the sampling distribution of $\sup_{x\in\mathbb{R}} |\widetilde{F}_{\mathcal{X}} - F_0(x)|$ is known (for every n) and does not depend on the choice of F_0. Therefore, for the usual KS test, the value c_α can be determined for any choice of α. However, this theorem is not applicable here, because the hypothesis H_0 is *composite*, it contains a whole family of distributions and not a single target distribution F_0: therefore we cannot use the usual KS calibration for our test.

This can be better understood by the following remark: when we observe data issued from a given GM distribution $F \in \mathcal{F}^{\text{GM}}$, then the empirical distribution of the data will much better fit the *estimated* GM distribution $\widehat{F}^{\text{GM}}_{\mathcal{X}}$, than it will fit the original underlying GM distribution F. This is because the estimated distribution is influenced by the particular data we have at hand (this phenomenon is sometimes called *overfitting*). Therefore, the statistic $\|\widetilde{F}_{\mathcal{X}} - \widehat{F}^{\text{GM}}_{\mathcal{X}}\|_\infty$ will tend to take lower values than what would do the statistic $\|\widetilde{F}_{\mathcal{X}} - F\|_\infty$. In other words, if we used the usual KS calibration for our test statistic defined in Equ. (3), then we would reject the null hypothesis less often than we would have to, and the risk we would announce would be erroneous.

5.1.2 Bootstrap as a calibration tool (the KS-fit test)

This calibration problem of goodness-of-fit tests when parameters need to be estimated beforehand is known in the statistical community, and the solution generally adopted to overcome it is to rely on the *bootstrap methodology*. The idea of the bootstrap is to take advantage of the information contained in the data by *resampling it*, in order to hopefully estimate properly the sampling distribution of the test statistic. It is somehow a computer-intensive procedure (especially when the sample size is high), but it provides good results, even for small data size: the complete explanation and investigation of our bootstrap method is already presented [2], let us provide a summarised description here.

The goodness-of-fit procedure is the following:

(a) Estimate the presumed underlying GM distribution by $\widehat{F}^{\text{GM}}_{\mathcal{X}}$ from the original sample $\mathcal{X}$;

(b) Compute the corresponding test statistic $KS_{\mathcal{X}}$ defined in Equ. (3);

(c) Repeat for $i = 1, \ldots, N$ (with N at least[2] 200) the following steps:

- (i) Simulate a sample $\mathcal{X}^{(i)} = x_1^{(i)}, \ldots, x_n^{(i)}$ following the estimated GM distribution $\widehat{F}^{\text{GM}}_{\mathcal{X}}$;
- (ii) Compute the estimation $\widehat{F}^{\text{GM}}_{\mathcal{X}^{(i)}}$ based on the i-th bootstrap sample $\mathcal{X}^{(i)}$;
- (iii) Compute the KS distance $KS_{\mathcal{X}^{(i)}}$ (shortened as $KS^{(i)}$) between the empirical distribution $\widetilde{F}_{\mathcal{X}^{(i)}}$ of the i-th bootstrap sample, and the GM distribution $\widehat{F}^{\text{GM}}_{\mathcal{X}^{(i)}}$ estimated from it;

(d) Denote by $KS^{(1)} \leq \ldots \leq KS^{(N)}$ the N distances obtained in step (c), ordered from the smallest to the highest value. Then:

- (i) Define the critical value c_α as the value[3] $KS^{([N(1-\alpha)])}$;
- (ii) Define the p-value $p(\mathcal{X})$ associated with the data as the proportion of the N values $KS^{(1)} \leq \ldots \leq KS^{(N)}$ which exceeds the initial value $KS_{\mathcal{X}}$;

(e) Conclude as follows:

- reject H_0 in favour of H_1 at risk α if $KS_{\mathcal{X}} > c_\alpha$;
- or, equivalently reject H_0 in favour of H_1 at risk α if the p-value $p(\mathcal{X})$ is smaller than α.

It should be reminded here that, in statistics, the p-value of a statistical test is the minimum risk one can undertake when rejecting H_0 (by risk, we mean the risk of rejecting H_0 while in fact it holds true).

The idea behind this bootstrap methodology is the following. The N bootstrap samples have the same size as the original data, they are GM distributed with distribution as close as possible to that of the original data, and therefore the N distances $KS^{(1)} \leq \ldots \leq KS^{(N)}$ (obtained in step (c) above) provide a good idea of the distribution of the test statistic $KS_{\mathcal{X}}$ if the original data were indeed GM distributed. Therefore, if the original data do not fit well a GM distribution, then the observed value $KS_{\mathcal{X}}$ will be high with respect to the values $KS^{(1)} \leq \ldots \leq KS^{(N)}$, and consequently the p-value $p(\mathcal{X})$ will be low. On the other hand, if the original data indeed follows (or can be approached by) a GM distribution, then the initial value $KS_{\mathcal{X}}$ is likely to be not particularly high with respect to the values $KS^{(i)}$, and the p-value $p(\mathcal{X})$ will consequently be moderate or high, which means that we will not be able to reject H_0 (and in practice, we will consider the GM model as adequate for our data).

It is usually considered that this bootstrap methodology provides good estimates of the actual p-values of statistical tests in practice, even for n moderate or small. However, for-

2. $N = 200$ is a reasonable value but a higher value of 500 can be taken for better estimation of the p-value, because we are studying the tail distribution of the test statistic, and not just estimating one of its central parameters

3. where $[x]$ denotes the integer part of x

mal proofs are hard to obtain, particularly for complicated models such as the mixture models (because there are no closed formulas for the estimators of the numerous parameters of the model), and the only existing formal proofs found in the literature are only asymptotically valid (*i.e.* when the sample size n is very high). We thus validated our method by relying on exhaustive simulations [2]. Note in addition that our study [2] also contains empirical demonstrations that our bootstrap calibrated goodness-of-fit test also has good *power*, in the statistical sense: this means that, in the presence of data which cannot be reasonably modelled with a Gaussian mixture, our test has the ability of detecting it with good probability (even for small or moderate values of the sample size n).

After building a GM model for describing the performances of a given version of program, we aim at using parametric statistics for comparing between the performances of distinct program versions. For this purpose, we propose and study new performance metrics in the next section.

6 New program performance metrics

In the literature, people are mainly focused on the average or median program performance. However in practice, the average or the median value may not be the most interesting summary measure that reflects the program performances, or may not be the best performance metric to use for selecting the most suitable program version.

For instance, consider the situation of a very long running application that a user executes very few times. The user has the choice between many code versions, which one should he select? If he bases his choice on the expected average or median execution time only, he may be disappointed if he executes his application very few times. If an application is rarely executed, the mean or median performances are not felt relevant, especially if the performance distribution is multi-modal. So, additional performance metrics can help him making a better selection.

Also, consider the situation where a user wants to know if the performances of his application are stable or not. Which metric can he use? The well known variance is a metric that measures how data is spread out around the average only: knowing how to interpret this metric is not so widespread among the end-users, and can be misleading when the data distribution is multi modal, because the variance is a measure of dispersion around a single value, the average. Therefore it cannot be the unique metric used for evaluating performance stability, additional metrics can be introduced and used, especially when the data are clustered as we will see later.

This section provides new performance metrics that help the user to select a *good* program version based on performance analysis. Let X and Y be two random variables, representing the performances of two code versions. Let $\mathcal{X}$ be a sample of X and $\mathcal{Y}$ be a sample of Y, meaning that $\mathcal{X} = (x_1, \ldots, x_n)$ and $\mathcal{Y} = (y_1, \ldots, y_m)$, with n and m denoting the respective sample sizes. Below we present four new performance metrics.

6.1 The metric $\mathcal{I}_1$: the mean difference

We may be interested in quantifying the average difference between the performances of two code versions. That is, we may be interested in computing the expected value $\mathbb{E}[\,|X-Y|\,]$. This defines our first performance metric as $\mathcal{I}_1 = \mathbb{E}[\,|X-Y|\,]$. Our parametric estimation of $\mathcal{I}_1$, noted $\widehat{\mathcal{I}_1}$, assumes that both X and Y are modelled with GM distributions. This means that the underlying p.d.f. f_X and f_Y of X and Y equal weighted combinations of Gaussian p.d.f. $f_X(x) = \sum_{i=1}^{K} \pi_i \varphi(x;\mu_i;\sigma_i)$ and $f_Y(x) = \sum_{j=1}^{K'} \pi'_j \varphi(x;\mu'_j;\sigma'_j)$. We recall here that $\varphi(x;\mu;\sigma)$ denotes the p.d.f. of the Gaussian distribution $\mathcal{N}(\mu,\sigma)$, and K and K' are the respective numbers of clusters of these Gaussian mixtures. Under this model, we readily have

$$\mathcal{I}_1 = \iint |x-y|\, f_X(x) f_Y(y)\, dx\, dy$$

$$= \sum_{i=1}^{K}\sum_{j=1}^{K'} \pi_i \pi'_j \iint |x-y| \varphi(x;\mu_i;\sigma_i)\, \varphi(y;\mu'_j;\sigma'_j)\, dx\, dy$$

$$= \sum_{i=1}^{K}\sum_{j=1}^{K'} \pi_i \pi'_j \mathbb{E}\left[\,|Z_i - Z'_j|\,\right]$$

where Z_i and Z'_j denote independent Gaussian variables with distributions $\mathcal{N}(\mu_i,\sigma_i)$ and $\mathcal{N}(\mu'_j,\sigma'_j)$. By classical properties of the Gaussian family, $Z_i - Z'_j$ has distribution $\mathcal{N}\left(\mu_i - \mu'_j, \sqrt{\sigma_i^2 + (\sigma'_j)^2}\right)$. Therefore, we use the following formula (proved [2]): if Z has distribution $\mathcal{N}(\mu,\sigma)$ then $\mathbb{E}[\,|Z|\,] = (2\Phi(\mu/\sigma) - 1)\mu + 2\sigma\varphi(\mu/\sigma)$. This entails the following formula for our theoretical performance metric:

$$\mathcal{I}_1 = \sum_{i=1}^{K}\sum_{j=1}^{K'} \pi_i \pi'_j \left((\mu_i - \mu'_j) \left(2\Phi\left(\frac{\mu_i - \mu'_j}{\sqrt{\sigma_i^2 + (\sigma'_j)^2}} \right) - 1 \right) + \sqrt{\frac{2(\sigma_i^2 + (\sigma'_j)^2)}{\pi}} e^{-(\mu_i - \mu'_j)^2/(2(\sigma_i^2 + (\sigma'_j)^2))} \right)$$

Consequently, using the estimations $\hat{\theta}, \hat{K}, \hat{\theta}', \hat{K}'$ of the parameters θ, K, θ', K' (*i.e.* the parameters of the estimated Gaussian mixtures distributions $\widehat{F}_{\mathcal{X}}^{\text{GM}}$ and $\widehat{F}_{\mathcal{Y}}^{\text{GM}}$), the parametric estimation $\widehat{\mathcal{I}_1}$ of our first performance metric $\mathcal{I}_1$ comes:

$$\widehat{\mathcal{I}_1} = \sum_{i=1}^{\hat{K}}\sum_{j=1}^{\hat{K}'} \hat{\pi}_i \hat{\pi}'_j \left((\hat{\mu}_i - \hat{\mu}'_j) \left(2\Phi\left(\frac{\hat{\mu}_i - \hat{\mu}'_j}{\sqrt{\hat{\sigma}_i^2 + (\hat{\sigma}'_j)^2)}} \right) - 1 \right) + \sqrt{\frac{2(\hat{\sigma}_i^2 + (\hat{\sigma}'_j)^2)}{\pi}} e^{-(\hat{\mu}_i - \hat{\mu}'_j)^2/(2(\hat{\sigma}_i^2 + (\hat{\sigma}'_j)^2))} \right)$$

6.2 The metric $\mathcal{I}_2$: the probability that a single program run is better than another

When the user needs to select which code version to execute, he may base his selection criteria on the expected average speedup for instance. But if his application is rarely executed, the average performance gain may not be interesting for him. He may be interested in executing a single time his application, and he wishes that a single run

has the best chances of being the fastest between two code versions. Formally, to help the decision, we can compute $\mathbb{P}[X < Y]$, the probability that a single run of X would be better than a single run of Y. This defines our second metric of program performances $\mathcal{I}_2 = \mathbb{P}[X < Y]$.

Our parametric estimation assumes that both X and Y are modeled with Gaussian mixture distributions.

$$\begin{aligned}\mathcal{I}_2 &= \iint \mathbb{1}_{x<y}\, f_X(x) f_Y(y)\, dx\, dy \\ &= \sum_{i=1}^{K}\sum_{j=1}^{K'} \pi_i \pi'_j \iint \mathbb{1}_{x<y}\, \varphi(x;\mu_i;\sigma_i)\, \varphi(y;\mu'_j;\sigma'_j)\, dx\, dy \\ &= \sum_{i=1}^{K}\sum_{j=1}^{K'} \pi_i \pi'_j \mathbb{P}\left[Z_i - Z'_j < 0\right]\end{aligned}$$

where Z_i and Z'_j denote independent Gaussian variables such that $Z_i - Z'_j$ is Gaussian distributed with expectation $\mu_i - \mu'_j$ and variance $\sigma_i^2 + (\sigma'_j)^2$. Since $\mathbb{P}[Z < 0] = \Phi(-\mu/\sigma)$ whenever Z has distribution $\mathcal{N}(\mu, \sigma)$, we thus have $\mathcal{I}_2 = \sum_{i=1}^{K}\sum_{j=1}^{K'} \pi_i \pi'_j\ \Phi\left(\frac{\mu'_j - \mu_i}{\sqrt{\sigma_i^2 + (\sigma'_j)^2}}\right)$. Consequently, plugging in the estimators of the parameters of the Gaussian mixture distributions leads to the following parametric estimator of $\mathcal{I}_2$:

$$\widehat{\mathcal{I}_2} = \sum_{i=1}^{\hat{K}}\sum_{j=1}^{\hat{K}'} \hat{\pi}_i \hat{\pi}'_j\ \Phi\left(\frac{\hat{\mu}'_j - \hat{\mu}_i}{\sqrt{\hat{\sigma}_i^2 + (\hat{\sigma}'_j)^2}}\right)$$

Alternatively, we can also generalise this metric to consider a constant real shift $\Delta \in \mathbb{R}$ to check between X and Y, and thus consider $\mathcal{I}_2 = \mathbb{P}[X < Y + \Delta]$ with its parametric estimation:

$$\widehat{\mathcal{I}_2} = \sum_{i=1}^{\hat{K}}\sum_{j=1}^{\hat{K}'} \hat{\pi}_i \hat{\pi}'_j\ \Phi\left(\frac{\Delta + \hat{\mu}'_j - \hat{\mu}_i}{\sqrt{\hat{\sigma}_i^2 + (\hat{\sigma}'_j)^2}}\right)$$

6.3 The metric $\mathcal{I}_3$: the probability that a single run is better than all the others

In practice, a user may have more than only two code versions. How can he select the best code version among many others, for a single run only ? Comparing code versions two by two is unfortunately misleading. All code versions must be compared together, not two by two. Let $X_1, X_2, \cdots, X_r$ denote r random variables corresponding to r distinct code versions. These r distinct random variables must not be confused with a sample $(X_1, X_2, \cdots, X_n)$ of a single random variable X. We propose to compute the probability that one code version, say the first one, executes faster than all the others for a single run only (not in average or in median), *i.e.* we compute the probability that $X_1 < \min(X_2, \ldots, X_r)$. This defines the following program performance metric:

$$\mathcal{I}_3 = \mathbb{P}[X_1 < \min(X_2, \cdots, X_r)]$$

The parametric estimation of $\mathcal{I}_3$ is noted $\widehat{\mathcal{I}_3}$. Here we assume that, for every given $j \in \{1, \ldots, r\}$, the random variable X_j is distributed as a Gaussian mixture with parameters $K = K_j$ and $\theta = \theta_j = (\pi_{1,j}, \ldots, \pi_{K_j,j}; \mu_{1,j}, \ldots, \mu_{K_j,j}; \sigma_{1,j}, \ldots, \sigma_{K_j,j})$.

As we did for $\mathcal{I}_1$ and $\mathcal{I}_2$, we need to obtain a formula for the metric $\mathcal{I}_3$ in terms of the parameters. Let us note $Y = \min(X_2, \cdots, X_r)$, and let G be the c.d.f. of Y. By the mutual independence of $X_1, X_2, \ldots, X_r$, the variables X_1 and Y are independent and therefore a classical probability property yields, since $\mathbb{P}[x < Y] = (1 - G)(x)$,

$$\begin{aligned}\mathcal{I}_3 &= \mathbb{P}[\, X_1 < Y\,] \\ &= \mathbb{E}[(1 - G)(X_1)] = \int_{-\infty}^{\infty} (1 - G(x)) f_1(x) dx.\end{aligned}$$

By independence of the variables $X_2, \ldots, X_r$, and Equ. (2), we have:

$$(1 - G)(x) = \mathbb{P}[\, X_2 > x, \ldots, X_r > x\,] = \prod_{j=2}^{r} \mathbb{P}[X_j > x]$$

$$= \prod_{j=2}^{r} \sum_{i=1}^{K_j} \pi_{i,j}(1 - \Phi(x; \mu_{i,j}; \sigma_{i,j})).$$

Our parametric estimator $\widehat{\mathcal{I}_3}$ of the metric $\mathcal{I}_3$ is then equal to the following integral (which we compute numerically, by using the R software for instance)

$$\widehat{\mathcal{I}_3} = \int_{-\infty}^{\infty} (1 - \hat{G}(x)) \hat{f}_1(x) dx$$

where $\hat{G}(x) = \prod_{j=2}^{r} \sum_{i=1}^{\hat{K}_j} \hat{\pi}_{i,j}(1 - \Phi(x; \hat{\mu}_{i,j}; \hat{\sigma}_{i,j}))$ and $\hat{f}_1(x) = \sum_{i=1}^{\hat{K}_1} \hat{\pi}_{i,1} \varphi(x; \hat{\mu}_{i,1}; \hat{\sigma}_{i,1})$

6.4 The metric $\mathcal{I}_4$: the variability level

People do not always know how to quantify the variability of programs performances. By default, they use the variance, but they may not know how to interpret it correctly. The variance measures how the data spread out around the average: but if the data are multi-modal or present clusters, then the average is not necessarily a good measure of the variability, especially when the modes or clusters are particularly distant from each other, and therefore the variance loses its attractiveness.

We propose to consider an alternative or complementary measure of the variability of programs performances: the number of modes of the underlying p.d.f. of the data, which we will note $\mathcal{I}_4$. It is simply equal to the number of *local maxima* of the p.d.f., which is supposed to represent the different values around which the data are spreading or clustering. A local maxima is called a *mode* in statistics.

Thanks to the Gaussian mixture modelling, which yields an explicit formula for the estimated p.d.f., we can compute a parametric estimation $\widehat{\mathcal{I}_4}$ of $\mathcal{I}_4$, which is simply equal to the number of local maxima of the Gaussian mixture p.d.f. $f_{\hat{\theta},\hat{K}}$ estimated from the data. Often, this estimation $\widehat{\mathcal{I}_4}$ turns out to be equal to $\hat{K}$, the estimated number of clusters of the fitted Gaussian mixture distribution. But it is not necessarily always the case, since sometimes the Gaussian mixture fitting algorithm proposes a higher value of $\hat{K}$ than the actual number of groups in the data, in order

to flexibly account for assymetry. Formally, the variability level is computed as:

$\widehat{\mathcal{I}_4}$ = the number of local maxima of the estimated Gaussian mixture p.d.f.

7 Implementation and Experiments

Because of paper size limitation, and for the sake of synthesis, this section provides the most important experimental results. The complete empirical study is presented in the research report [2].

7.1 Implementation using R statistical framework

All the statistical methods, KS test, metrics and procedures presented in this article have been implemented in R. The source code and few data sample are publicly available [2].

Regarding the algorithm of clustering (EM and choice of K) presented in Sect. 4, it has been implemented in a great variety of languages and statistical softwares. Since we intend to use the R statistical framework, we used some existing packages. We chose to use the popular `Mclust` package [7]. There exists other packages doing the job (like `Rmixmod` or `EMCluster` for instance). We use `Mclust` because it is efficient, widely used, documented, and has the advantage of handling both common variances model and distinct variances model: it can therefore always propose an estimated model, even in situations where the distinct variances model causes problems (as a matter of fact, when all the σ_k are supposed equal, the likelihood function $L(\theta)$ always has a maximum, which is not always the case when $\sigma_1, \ldots, \sigma_K$ are estimated separately).

7.2 Empirical validation of the KS fitting test

In Sect. 5, we described the statistical procedure which makes it possible to test the hypothesis that some given dataset is issued from some Gaussian mixture distribution. This goodness-of-fit test is calibrated via bootstrap, and the procedure needs to be validated via some simulations (because formal proofs are not manageable, for finite sample sizes n) in order to assess the following:

- (i) *the bootstrap calibration yields the announced risk for the test*, *i.e.* whatever α may be chosen, if we apply the test to a dataset satisfying H_0 (*i.e.* issued from a GM distribution) with a critical value which is supposed to be associated with the risk α, then there is indeed a probability α (or very close to α) that the test leads to a rejection of H_0.
- (ii) *the constructed test has satisfying power*, *i.e.* if we apply the test to a dataset which is not issued from a GM distribution, then the test will lead to a rejection with sufficient probability. This should be as high as possible when the true underlying distribution is very different from a GM, for instance when the data exhibit a neat/strong asymmetry (in the right tail for instance, but not exclusively), or heavy tails (ie presence of several extreme values).

The first objective (i) is clearly defined and the way it should be investigated via simulation is clear as well. The second objective (ii) is vaguer, and consequently we were partially able to verify it by simulation (because, on one hand, there is an infinity of distributions which are not GM, and with different intensity, and on the other hand the definition of a satisfying power cannot be clearly set non-asymptotically). Both the objectives have been studied and validated by extensive simulations with $N \in \{500, 1000\}$ and $n \in \{30, 100, 500\}$. The reader is invited to study Sect. 5.2 of the report [2] for more details about QQ-plot analysis of the estimated KS statistic .

7.3 Experimental results on clustering

We collected during more than 10 years a great number of performance data samples, resulting from many empirical studies: SPEC CPU applications (2001, 2006), all SPEC OMP applications, NAS Parallel Benchmark, own micro-benchmarks, other parallel applications, various compilers versions and options, Linux versions, different HPC machines architectures and generations, etc. The number of samples is 2438, each one contains between 30 (the majority of samples) and 1000 execution times. Our data are very representative of the diversity of what is commonly experimented in the HPC and intensive computation communities. Note that, while we consider execution times as example of study, any continuous performance data can be analysed using our statistical method (energy consumption, network traffic, input/output latency, throughput, etc.).

Our clustering method was applied on each sample with success. The computation time of the clustering method using R is fast, in almost all cases it does not exceed few milliseconds per sample. When a clustering is applied on a sample, it computes a Gaussian mixture model. That is, for each sample, a number $K \in \mathbb{N}$ of clusters is computed (it is the chosen number of Gaussian components selected by the BIC criterion described in Sect. 4). Regarding the obtained number of clusters, the median value is equal to 2, which means that at least half of the samples can be modelled with mixtures of 2 Gaussians, and half of the samples can be modelled with mixtures of more than 2 Gaussians. The third quartile is equal to 3, which means that at least 75% of the samples are modelled with mixtures of 1, 2 or 3 Gaussians. The maximal observed number of clusters is 9, which means that there are samples which required a model with 9 components (clusters).

7.4 Experiments on data-model fitting

In this section, we study the quality of data-model fitting. After the computation of an estimated Gaussian mixture distribution, our KS-fit method tests the quality of the fitting between the Gaussian mixture model and the data as explained Sect. 5. It returns a probability called p-value: remind that this probability is the risk of error when falsely rejecting the hypothesis that the data is not issued from a Gaussian mixture model. Therefore, if the p-value is low, then we reject the fitting between the data and the Gaussian mixture model. If it is not, then the model is acceptable for the data at hand.

Fig. 5 illustrates the histogram of the obtained p-values: clearly, we observe that the Gaussian mixture is a very good modelling for the majority of the samples. Indeed, if we consider a given risk value $0 < \alpha < 1$ of, say, 5%, we

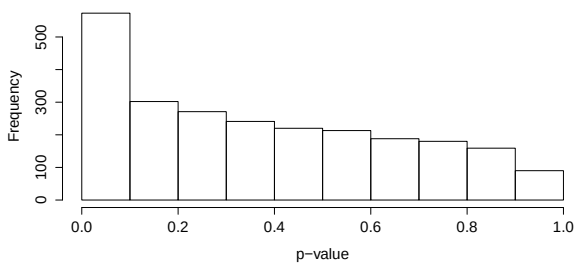

Figure 5: Histogram of the p-values.

count the proportion of samples having a p-value smaller or greater than 5% (meaning that the Gaussian mixture model is rejected at this risk), and we respectively find 16.81% and 83.09%. This means that the Gaussian mixture model is an acceptable model for 83% of the samples, which is quite high and satisfying[4].

We investigated the reasons why some samples did not fit the Gaussian mixture model (16.81% of the samples). We found out that there are basically two main reasons:

1) Some samples are apparently issued from distributions which cannot easily be approximated by Gaussian mixtures. For instance, heavy-tail distributions, exponential distributions, Pareto distributions, etc. The rejection of the Gaussian mixture model is therefore logical, since this model is flexible, but not that flexible.
2) In the majority of the situations though, the reason is that samples contain *ties* (data values which are repeated inside the same sample). As a matter of fact, our experiments collected execution times with some rounding precision: it may thus happen that some execution times are artificially perfectly equal in the sample. And when a too high proportion of data values are tied, our fitting test turns out to be *severe*, because it is not designed to be applied to non-continuous data, and ties artificially increases the Kolmogorov-Smirnov distance (with "high steps" in the step function $\widetilde{F}_\chi$). We strongly believe that, in practice, we can reduce the severity of our test by increasing the precision of the collected data, and find out that the Gaussian mixture model suits even more situations that it seems in Fig. 5.

In regard to the great variety of real data samples we have at our disposal, we can conclude that Gaussian mixtures are a good and flexible model for describing program execution times in general. Furthermore, they are good candidates for modelling any other kind of continuous performances (energy consumption, network traffic, memory bandwidth, etc.).

7.5 Empirical validation of the precision of our performance metrics

We also undertook an extensive set of experiments to check the accuracy of our new performance metrics $\widehat{\mathcal{I}_1}$, $\widehat{\mathcal{I}_2}$, $\widehat{\mathcal{I}_3}$ and $\widehat{\mathcal{I}_4}$ compared to their theoretical values $\mathcal{I}_1$, $\mathcal{I}_2$, $\mathcal{I}_3$ and $\mathcal{I}_4$. We used simulations to generate various random Gaussian mixture models , with diverse parameters, to have a a big number of theoretical values of these metrics ($N = 1000$, $n \in \{30, 100, 200\}$, $r = 5$). Then, for each theoretical GM model, we constructed an estimated GM model using our clustering method. Then we computed and analysed the mean square errors and mean absolute percentage errors of $\widehat{\mathcal{I}_1}$, $\widehat{\mathcal{I}_2}$, $\widehat{\mathcal{I}_3}$ and $\widehat{\mathcal{I}_4}$, with respect to the theoretical exact values.

The details of our experiments and results can be checked in Sect. 6.5 of the report [2]. We concluded there that our parametric estimations of our new performance metrics are sufficiently accurate so that they can be used in practice with confidence.

7.6 On the variability of program execution times

Based on our metric for the estimation of the variability level ($\mathcal{I}_4$ defined in Sect. 6.4), we computed the estimated values of this metric for all samples. Fig. 6 plots the histogram of the obtained variability levels. In theory, recall that the variability level is not necessarily equal to the number of clusters (Gaussians) of the GM model. We observed the following:

- $\approx$ 37% of the samples have a variability level equal to one, which means that the execution times are spread around a single value.
- $\approx$ 32% of the samples have a variability level equal to 2, which means that the execution times are spread around two values.
- $\approx$ 12% of the samples have a variability level equal to 3, which means that the execution times are spread around three values.
- $\approx$ 19% of the samples have a variability level ≥ 4, which means that the execution times are spread around more than three values.

This histogram clearly demonstrate that summarising the execution times of a program with a single number (mean or median) is often inadequate.

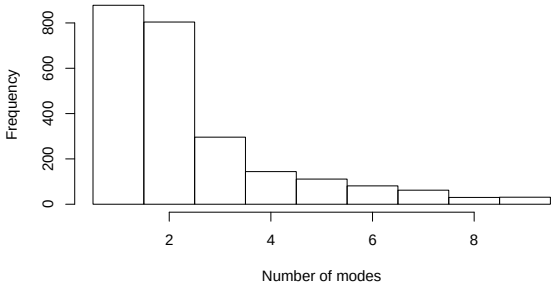

Figure 6: Variability levels of programs executions times

The next section presents some state of the art in code performance analysis using statistics.

4. We note here that we did additional study to test the modelling of the data with Lognormal or Weibull distributions, we figured out that these unimodal distributions are not satisfactory, GM distributions clearly exhibit better fitting.

8 Related work

8.1 Observing execution times variability

The literature contains some experimental research highlighting that program execution times are sometimes increasingly variable or unstable. In the article of *raced profiles* [8], the performance optimisation system is based on observing the execution times of code fractions (functions, and so on). The mean execution time of such code fraction is analysed thanks to the Student's t-test, aiming to compute a confidence interval for the mean. This previous article does not fix the data input of each code fraction: indeed, the variability of execution times when the data input varies cannot be analysed with the Student's t-test. Simply because when data input varies, the execution time varies inherently based on the algorithmic complexity, and not on the structural hazards. So assuming that execution times vary in this situation according exclusively to hazard is an obviously wrong approach.

Program execution times variability has been shown to lead to wrong conclusions if some execution environment parameters are not kept under control [9]. For instance, the experiments on sequential applications [9] show that the size of Unix shell variables and the linking order of object codes both may influence the execution times. However, it should be noted that one of the experimented benchmark (`perlbench`) has a hidden input which is an environment variable. So if the environment variable size varies, this means that the program input varies, so it is expected that executions times vary as consequence. Thus, the variability here cannot be considered as randomness.

An empirical study of performance variation in real world benchmarks with fixed data input has been published [3]. Our study concludes three points: 1) The variability of execution times of long running sequential applications (SPEC CPU 2000 and 2006) can be marginal if we fully control the hardware machine. 2) The variability of execution times of long running parallel applications such as SPEC OMP 2001 is important on multicore processors, such variability cannot be neglected. 3) Almost all the samples of execution times do not follow a Gaussian distribution.

In the current research study, the variability of code performance is not related to varying data input, but is related to external factors of the binary code that the user cannot control.

8.2 Program performance evaluation in presence of variability

Program performance analysis and optimisation may rely on two well known books that explain digest statistics to our community [10], [11] in an accessible way. These two books are good introductions for doing fair statistics for analysing performance data. Based on these two books, previous work on statistical program performance evaluation have been published [12]. In the latter article, the authors rely on the Student's t-test to compare between two average execution times (the two sided version of the student t-test) in order to test whether two theoretical means are equal or not. We improved this previous work [1]: first, we showed how to conduct a *one-sided* Student's t-test to validate that $\mu_X > \mu_Y$. Second, we showed how to check the normality in small samples and the equivalence of their variances (using the Fisher's F-test) in order to use the classical Student's t-test instead of the Welch's variant.

In addition, we must note that the known books [10], [11], [12] focus on comparing between the mean execution times only. When the program performances have some extrema values (outliers), the mean is not a good performance measure (since the mean is sensitive to outliers). Consequently the median is usually advised for reporting performance indicators (such as for SPEC scores). Consequently, we relied on well known academic books in statistics [13], [14], [15] for comparing between two medians in the speedup-test protocole [1].

8.3 References on Gaussian mixtures, goodness-of-fit and bootstrap

References about mixture models and particularly Gaussian mixture models are numerous in the statistics literature, machine-learning literature, as well as in many statistics-using fields (in particular image analysis, bioinformatics, biology, medicine, etc). We will therefore only cite the reference book on mixture models [5], which is an excellent starting point for the theory and a great source of applications of the subject. Other possible general textbooks on mixtures exist too [16], [17]. Note that in many settings, the concern is about mixtures of *multivariate* Gaussian distributions: our univariate framework is a subcase, and therefore several issues evoked in the clustering literature are therefore not relevant to our framework.

The use of the EM algorithm as a solution to finite mixtures fitting is a classical subject in the statistics and pattern recognition literature, since the release of the breakthrough paper [4]. In the present work, we decided to address the important issue of choosing the appropriate number K of components by relying on the BIC criterion: there are however several popular alternatives [18], [19].

Well known references about goodness-of-fit tests are the books [20], [21] for instance. However, this topic is often addressed in standard textbooks of advanced statistical analysis also. Chapter 19 of the book [6] is one reference concerning the problem of calibrating the Kolmogorov-Smirnov test when the target is a whole family of distributions (but it stays on the theoretical ground).

General references about the bootstrap and its use in applied statistics are the following books [22], [23], [24]. Some specific references about the use of the bootstrap principle for addressing the problem of calibration in goodness-of-fit tests are the articles [25], [26], [27]. We point out that these existing work only provide theoretical results which guarantee the asymptotic (*i.e.* for large n) validity of the bootstrap principle, and would be applicable in our framework only for a fixed number K of components (they also require asymptotic normality results about the estimator $\hat{\theta}$, which we cannot formally guarantee). These important work should thus be considered as very good signals, but simulation experiences were quite necessary in the present work to validate our method of Gaussian mixtures fitting test.

9 Perspectives and discussions

9.1 Multi-dimensional performance modelling

Nowadays, people are not only interested in analysing and optimising a single type of program performance. Users can be interested in studying multiple types of performances conjointly: execution time, energy consumption, power consumption, memory consumption, network traffic, memory-CPU bandwidth, input/output latency, etc. This means that we must be able to collect and measure performance data conjointly, saved in multi-dimensional data.

As far as we know, no statistical study on multi-dimensional performance has been done. The aim here would not be to write a model of a single performance dimension as a function of the other performance types, as is done with linear and non-linear regression models. The objective would be to analyse the correlation and the relationship between performances of different natures. Regression models are not suitable because they make a projection of a performance dimension, and do not model all performance dimensions conjointly. Fortunately, Gaussian mixtures naturally and easily adapt to the multidimensional framework (see [5] for examples of applications of mixtures of multivariate Gaussians): we could thus be in a position of modelling, for instance, the bi-dimensional observed performances {execution time, energy consumption} as a mixture of bi-dimensional Gaussian distributions, which would result in exploring the possible relationship between the time performance and the consumption performance, more precisely than by just computing the coefficient of correlation between these two indicators (which is a global, and sometimes misleading, indicator of statistical relationship). This could be a good starting point for investigating the possible hardware and software reasons that lead to a variability of multimodal type.

9.2 Considering mixtures of other distributions

Gaussian mixture modelling is a natural and good method for fitting multi-modal distributions. According to our experiments, Gaussian mixtures fit most of the cases (execution times in our situation). However, some cases cannot be modelled by GM. If the observed data are issued from heavy tail distributions, exponential distributions, Pareto distributions, or mixtures of these distributions, then the Gaussian mixture modelling may be disappointing: in that case, mixtures of other families of distributions (other than the Gaussian family) could be considered. This should be considered as necessary only in particular situations though.

Another disadvantage of GM model is that it considers theoretical performance values from $-\infty$ to $+\infty$. In practice, if we assume that a program executes and terminates correctly, the theoretical performance values are bounded. So GM mixture are not necessarily well suited for studying extreme values such as minimal execution times and worst case execution times. For extreme values statistics, other data distributions should be used.

9.3 Discussion: how to decide about the best code version ?

When dealing with efficient programming, an application may have different code versions: multiple source code versions may exist, also with multiple binary code versions (various compilation flags). The question of selecting the most efficient code version on a given machine, for a given data input set, under a given software environment, is not easy. Statistics provide a tool to help decision, statistics do not provide strict guarantees, since the conclusions issued from statistical methods always come along with an error factor, a risk which is often only proved asymptotically. Fortunately in practice, this computed risk may turn out to be quite fair/accurate even for moderate or small sample sizes, as demonstrated in our situation by our experimental study.

For helping decision making, suitable performance metrics must be considered:

- If the code is devoted to be run a high number of times, it may be suitable to chose a code version with the best mean or median performance. The median performance is more suitable for codes that exhibit outliers.
- If a code is not executed a high number of times, it would better to study the probability that a single run of a code version would be better than a single run of another code version or other multiple code versions. We proposed in Sect. 6 such performance metrics.

10 Conclusion

When executing a binary code on a physical machine, one could be interested in analysing and optimising the performance. The performance considered in our work is any continuous value, such as execution time, energy consumption, power consumption, network traffic, etc. In ideal execution environment, when a user has a full control on the executing machine and operating systems, it may be possible to stabilise program performance. But in practice, users do not have full control on their machine and operating systems. On realworld executing machines, the sharing of resources and the modern processor micro-architectures make it hard to observe stable performance. And the observed performances are quite rarely normally distributed, although many people consider or expect it is. Indeed, we observed that the collected performances are multi-modal distributions.

Gaussian mixtures seem to be good model for the majority of the performances that we observed in practice. Also, such data distributions provide interesting perspective regarding multi-dimensional performance data. Indeed, an interesting performance analysis must consider the performance as multi-dimensional data, each dimension corresponds to a specific performance nature. For instance we can consider the triplet {Execution time, energy consumption, network traffic}. Fortunately Gaussian mixtures are a good solution for modelling multi-dimensional data in order to analyse the relationship between multiple dimensions.

Concerning the performance of the goodness-of-fit test we introduced in this work, we found out that it was very satisfying. We nonetheless observed that, in the presence of too much a proportion of equal data in the sample (ties), our fitting test tends to artificially reject the Gaussian mixture model too often: a simple solution to this problem would be to increase the precision of the measurement method so

that the risk of observing tied values (exæquo) is severely reduced.

Some people are interested in extreme values statistics (Worst Case Execution Times, Best Case Execution Times). In that case, Gaussian mixture modelling is not the adequate way of addressing this issue: either mixtures of other data distributions must be considered, or alternative methods should be considered (extreme value analysis techniques in particular, although they require quite a large number of data values to be truly reliable).

Our Gaussian mixture modelling provides a new and interesting metric to evaluate the variability of performance. Indeed, instead of only considering means and variances (the variances being, by the way, difficult to interpret in practice), we propose to consider as well the *modes* of the data distributions. Thus, the variability level of performances can be measured by the number of these modes, which we can compute with a parametric method based on Gaussian mixtures. The number of modes is a natural way of giving an idea of the performance variability: a data distribution with a single mode means that the performances are quite stable around a single value; with two modes, it means that the performances are varying around two values, etc. In addition, if the number of modes is greater than one, this can be a good indication of being careful with the interpretation (and comparison) of the variance (since the usual interpretation generally assumes that the data are issued from a unimodal distribution). Moreover, the existence of these modes can be further investigated by trying to explain them with auxiliary measurements made during the execution of the program, which is certainly the most fruitful perspective of this research work.

REFERENCES

[1] S. Touati, J. Worms, and S. Briais, "The Speedup-Test: A Statistical Methodology for Program Speedup Analysis and Computation," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 10, pp. 1410–1426, 2013. [Online]. Available: https://hal.inria.fr/hal-00764454

[2] J. Worms and S. Touati, "Parametric and Non-Parametric Statistics for Program Performance Analysis and Comparison," INRIA Sophia Antipolis - I3S ; Université Nice Sophia Antipolis ; Université Versailles Saint Quentin en Yvelines ; Laboratoire de mathématiques de Versailles, Research Report RR-8875, Mar. 2016. [Online]. Available: https://hal.inria.fr/hal-01286112

[3] A. Mazouz, S. Touati, and D. Barthou, "Study of Variations of Native Program Execution Times on Multi-Core Architectures," in *International Workshop on Multi-Core Computing Systems (MuCoCoS)*. Krakow, Poland: IEEE, Feb. 2010.

[4] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, no. 1, pp. 1–38, 1977.

[5] G. MacLachlan and D. Peel, *Finite Mixture Models*. New York: Wiley series in Probability and Statistics, 2000.

[6] A. W. van der Vaart, *Asymptotic statistics*. Cambridge University Press, 2000, iSBN-13: 978-0-521-784504.

[7] C. Fraley, A. Raftery, B. Murphy, and L. Scrucca, "mclust version 4 for r : Normal mixture modeling for model-based clustering, classification, and density estimation," Department of Statistics, University of Washington, Tech. Rep. 597, 2012.

[8] H. Leather, M. O'Boyle, and B. Worton, "Raced Profiles: Efficient Selection of Competing Compiler Optimizations," in *Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '09)*. ACM SIGPLAN/SIGBED, Jun. 2009.

[9] T. Mytkowicz, A. Diwan, P. F. Sweeney, and M. Hauswirth, "Producing wrong data without doing anything obviously wrong!" in *ASPLOS*, 2009.

[10] R. Jain, *The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modelling*. New York: John Wiley and Sons, inc., 1991, ISBN-13: 978-0-471-503361.

[11] D. J. Lilja, *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, 2000, ISBN-13: 978-0521641050.

[12] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous Java performance evaluation," in *Proceedings of the Twenty-Second ACM SIGPLAN Conference on OOPSLA*, ser. ACM SIGPLAN Notices 42(10), Montréal, Canada, Oct. 2007, pp. 57–76.

[13] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*. Springer, 2002, ISBN-13: 978-0387953519.

[14] M. Hollander and D. A. Wolfe, *Nonparametric Statistical Methods*. Wiley-Interscience, 1973, iSBN-13 978-0-471-190455.

[15] G. Saporta, *Probabilités, analyse des données et statistique*. Paris, France: Editions Technip, 1990, iSBN 978-2-7108-0814-5.

[16] D. M. Titterington, A. F. Smith, and U. E. Makov, *Statistical Analysis of Finite Mixture Distributions*. Wiley, 1985.

[17] K. L. Mengersen, C. P. Robert, and D. M. Titterington, *Mixture Estimation and Applications*. Wiley, 2011.

[18] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of clusters in a data set via the gap statistic," *Journal of the Royal Statistical Society, Series B*, vol. 63, no. 2, pp. 411–423, 2001.

[19] A. Fujita, D. Y. Takahashi, and A. G. Patriota, "A non-parametric method to estimate the number of clusters," *Computational Statistics and Data Analysis*, vol. 73, pp. 27–39, 2014.

[20] R. B. d'Agostino and M. A. Stephens, *Goodness-of-fit techniques*. CRC Press, 1986.

[21] O. Thas, *Comparing Distributions*. Springer, 2010.

[22] A. Davison and D. Hinkley, *Bootstrap Methods and their Application*. Cambridge Series in Statistical and Probabilistic Mathematics, 1997.

[23] M. R. Chernick, *Bootstrap Methods: A Guide for Practitioners and Researchers (2d edition)*. Wiley series in Probability and Statistics, 2007.

[24] B. F. Manly, *Randomization, Bootstrap and Monte Carlo Methods in Biology (2d edition)*. Chapman & Hall/CRC Press, 1997.

[25] W. Stute, W. G. Manteiga, and M. P. Quindimil, "Bootstrap based goodness-of-fit-tests," *Metrika*, vol. 40, pp. 242–256, 1993.

[26] G. J. Babu and C. R. Rao, "Goodness-of-fit tests when parameters are estimated," *Sankhyā : The Indian Journal of Statistics*, vol. 66, no. 1, pp. 63–74, 2004.

[27] G. J. Babu, "Resampling methods for model fitting and model selection," *Journal of Biopharmaceutical Statistics*, vol. 21, pp. 1177–1186, 2011.

Julien WORMS is a permanent assistant professor in Probability and Statistics, at the "Laboratoire de Mathématiques de Versailles" of the University of Versailles-St-Quentin (France). He works in Mathematical Statistics, especially on limit theorems for estimators or test statistics. His recent papers explore Extreme Value Analysis for incomplete time-to-event data.

Sid TOUATI is a permanent professor in the computer science department of Université Côte d'Azur (emerged from Université Nice Sophia-Antipolis, France). He is a member of I3S (CNRS) and INRIA-Sophia laboratories. He worked in advanced backend code optimisation for compilers. Currently he is interested in code performance improvement, evaluation and analysis for high performance applications on multi-core processors.

APPENDIX

Below, we give few technical examples of the use of our statistical method to model a dataset by a Gaussian mixture, to help making decision, to plot and print the result. Our software and data are freely available [2].

.1 Practical example 1 done with R

First, load the VARCORE software in R

```
> source('VARCORE.R')
```

Load a sample of data from a file, named C1, execution times in seconds

```
> C1 <- read.csv("ammp-C2", header=F)$V1
```

Clustering: building a Gaussian mixture model for the dataset C1.

```
> clusC1 <- VARCORE_Clustering(C1)
```

Plot the figure

```
> VARCORE_plot_clustering_result(clusC1,
        C1, maintitle="ammp",
        subtitle="Config 1",
        plot_legend=T,
        plot_rug=T,
        xlabel="Execution Times",
        ylabel="Probability")
> VARCORE_plot_clustering_result(clusC1,C1)
```

Extract cluster information and print them.

```
c = VARCORE_extract_cluster_information(clusC1)
c
```

Every line corresponds to a cluster (Gaussian) with its weight, mean and variance. Here we have 5 clusters.

```
     weights    means      stdevs
1 0.09677359 92.21333 0.163372026
2 0.15280620 93.26964 0.146702827
3 0.45830339 93.54552 0.227167773
4 0.16059283 94.21802 0.003986046
5 0.13152400 94.99607 0.392762363
```

Let us compute the variability level of C1.

```
> nbmodes = VARCORE_nbmodes_estimation(clusC1)
> nbmodes
[1] 4
```

So we say that the variability level of C1 is equal to 4, even if its number of clusters is equal to 5.

Extract the data that belong to every cluster (here we have 5 clusters)

```
> VARCORE_extract_cluster_list(clusC1,C1)
[[1]]
[1] 92.41 92.01 92.22

[[2]]
[1] 93.22 93.21 93.21 93.02 93.21

[[3]]
 [1] 93.61 93.62 94.01 93.42 93.82 93.41 93.61
      93.41 93.42 93.42  93.61 93.62 93.42 93.81 > p
[[4]]
[1] 94.22 94.22 94.22 94.22 94.21

[[5]]
[1] 95.61 95.02 94.62 94.81
```

Check the quality of the data fitting to the model. p-val gives the risk error: the higher it is, the better the fitting is.

```
> ksfit = VARCORE_calculKSFit(clusC1, C1)
> ksfit$pval
[1] 0.205
```

This means that the risk of rejecting (with error) the hypothesis that the data is issued from a Gaussian mixture model is 20.5%: 20.5% is not a small risk, so we do not reject the assumption that the data C1 fits well the computed Gaussian mixture model clusC1.

Plot fitting check.

```
> VARCORE_plotCDF_fit(clusC1, C1,
    maintitle="Estimated GM and empirical CDF",
    subtitle="Kolmogorov-Smirnov Fitting Test",
    xlabel="Data Values",
    ylabel="Probabilities")
```

.2 Practical example 2 done with R

Below, we give an example of using our software to compare the performance between two or multiple program versions.

Load three sets of data, corresponding to three program versions: C1, C2 and C3.

```
> C1 <- read.csv("ammp-C1", header=F)$V1
> C2 <- read.csv("ammp-C2", header=F)$V1
> C3 <- read.csv("ammp-C3", header=F)$V1
```

Start by building Gaussian mixture model for each of C1, C2 and C3.

```
> clusC1 <- VARCORE_Clustering(C1)
> clusC2 <- VARCORE_Clustering(C2)
> clusC3 <- VARCORE_Clustering(C3)
```

Estimate the probability that the execution time of a single run of C2 is lower than a single run of C3.

```
> p=VARCORE_probXY_Mclust(clusC2,clusC3)
> p
0.5092883
```

Estimate the probability that the execution time of a single run C1 is lower than a single run of C3.

```
> p=VARCORE_probXY_Mclust(clusC1,clusC3)
> p
0.9999989
```

Estimate the probability that $C2$ has the lowest execution time (the best code version), *i.e.* the execution time of $C2 = \min(C1, C2, C3)$

```
> p=VARCORE_probX1min_Mclust(
        list(clusC2, clusC1, clusC3))
```

```
[1]  2.891931e-07
```

Estimate the probability that $C1$ is has the lowest execution time (best code version), *i.e.* the execution time of $C1 = \min(C1, C2, C3)$

```
> p=VARCORE_probX1min_Mclust(
        list(clusC1, clusC2, clusC3))
> p
[1] 0.9999668
```

From above we can safely chose $C1$ as the code version that would certainly provide the lowest execution time for a single run only.

Compute the mean performance difference between C1 and C2

```
> val=VARCORE_meandiff_Mclust(clusC1,clusC2)
> val
[1] 7.507419
```

Compute the alpha-quantile of C1, for instance alpha=0.33.

```
> val=VARCORE_quantile_Mclust(0.33, clusC1)
> val
[1] 85.58795
```

From above we say that 33% of the runs of C1 would have an execution time below 85.58 seconds.

Compute the probability that a single run of C1 < 85 seconds.

```
> p=VARCORE_probXa(clusC1, 85)
> p
[1] 0.1874038
```

To conclude this example, the first version of the code (C1) is certainly always faster than the other two versions (and each of these two versions has about the same chance of being lower than the other), and it is estimated that about 33% of the time the first version has an execution time lower than 85.59 (and the probability that the execution time is lower than 85 drops to about 19%).